The XENIX® System V Development System

Release Notes

Version 2.1.3

The Santa Cruz Operation, Inc.

This document was typeset with an IMAGEN® 8/300 Laser Printer.

IMAGEN is a registered trademark of IMAGEN Corporation.
XENIX is a registered trademark of Microsoft Corporation.

Document Number: X86/286-6-20-86-3.0/2.1.3

Part Number: 014-030-003

Processed Date: Tue Jun 24 11:07:56 PDT 1986

XENIX System V Development System Release Notes

Release Notes
Release 2.1.3
XENIX®-86 System V for personal computers
XENIX-286 System V for personal computers
Development System
June 20, 1986

1. Preface

These notes pertain to the XENIX-86 and XENIX-286 Development System for personal computers. They contain notes on the software and documentation, and the procedure for installing the software.

We are always pleased to hear of user's experience with our product, and recommendations of how it can be made even more useful. All written suggestions are given serious consideration.

2. Software Notes

This release includes the Microsoft "cmerge C compiler." These notes will enable you to best take advantage of this compiler.

2.1 Include Files and Utilities

The machine dependent Development System *include* files and utilities are included on the XENIX version 2.1.3 Operating System N*n* Volumes. If you are using a 2.0 release of the XENIX Operating System with a 2.1.3 release of the XENIX Development System you will be missing **adb** and the */usr/include/sys* files.

If you are missing these files, contact the support department listed on the support information card included with the software, to request a floppy that includes these files.

2.2 cc(CP)

The cmerge compiler, **cc**(CP), supports several important features. It allows you to compile programs for both XENIX and DOS. It also has stronger typing restrictions than previous C compilers (for example, pointer arithmetic is more stringent).

2.2.1 Code Generation

The cmerge compiler, **cc**(CP), defaults to small model XENIX 8086 code generation. The default configuration flag is **−M0**.

The default stack for programs compiled on an 8086 machine is a variable stack, starting at the top of a full 64K byte data segment. A full 64K of physical memory is alloted for combined stack and data. The stack grows down until it hits the data. While variable stack wastes memory, it is useful for program development. Use the **−F** flag to specify a different fixed stack size. We recommend you use the **−F** flag on final versions for performance reasons.

The default stack for 286 machines is a fixed size stack of 2000 hexadecimal bytes. Use the **−F** flag to specify a different stack size. Variable stack size is not supported for 286 machines.

The libraries included in the SCO XENIX 3.0 Development Systems use big-endian word order. The libraries included in the XENIX System V Development System use little-endian word order. This refers to the order of bytes in long words.

The most important difference between compiling programs for XENIX and DOS is the libraries used.

The cmerge compiler is capable of producing binaries suitable only for 80286 computers and/or for DOS systems (by using the **−M2** or **−dos** flag). The default configuration uses only 8086 instructions and produces programs executable on 8086, 80186, and 80286 machines. Using the **−M2** flag causes 286 instructions to be used and produces programs executable only on 286 machines. It is possible to create programs on 8086 machines that can only be executed on 286 machines. We recommend using the default settings. Refer to the **machine**(M) manual page for information on binary compatibility.

2.2.2 DOS Development Files

DOS executable files are created by using the **–dos** flag with **cc**(CP). The **–dos** flag causes the compiler to use the DOS libraries and invoke the DOS linker, **dosld**(CP), instead of **ld**(CP). The DOS libraries are found in */usr/lib/dos*.

Refer to the XENIX *C Library Guide* (Appendix A "XENIX to MS-DOS: A Cross Development System," and **dosld**(CP)) for more information on using XENIX to create programs suitable for DOS systems. Also see section 4.2 "**cc**(CP)" in these *Release Notes* for information on using DOS floating-point.

2.2.3 Large Model Program Generation

This release supports large model 8086 or 80286 program generation for XENIX and DOS. Large model programs developed on 8086 machines under XENIX-86 will only run on 80286 machines under XENIX-286.

This release includes a fix to large model program generation. Previous versions of the compiler generated code that sometimes corrupted the swap map when a large text program did an exec. Sections of allocated swap space were either not freed or were assigned an incorrect starting block number. The effect was usually a decrease in the amount of swap space available, but swap i/o errors and overlap of swap space usage could also occur.

2.2.4 Large Model Passes of the Compiler

This release includes large model passes of the cmerge compiler. These are invoked using the **–LARGE** flag with **cc**. Only 286 machines can run the large model passes of the compiler. Refer to the **cc**(CP) manual page, and the chapter "Cc: A C Compiler" in the XENIX *C User's Guide* for more information on using large model passes.

If you encounter *segmentation violation* errors while using the large model passes of the compiler, do not use code optimization (**-O**). If the errors persist, use the standard compiler passes instead of the large model passes (do not specify **-LARGE** on the **cc** command line).

2.2.5 Huge Model Program Generation

Expressions similar to the following do not compile properly in huge model:

```
char *p, *foo;
if (--p < foo)
```

Use this type of construction instead:

```
--p;
if (p < foo)
```

2.2.6 Syntax - unsigned int

To specify an *unsigned int* use the following syntax:

```
typedef unsigned foo
```

instead of:

```
typedef unsigned int foo
```

2.2.7 Variable Declarations

A *Segmentation Violation: core dumped* error occurs if you declare too many variables on a single line. For example:

```
char *
        x1,     /* comment */
        x2,     /* more comment */
        .
        .
        .
        x934;   /* more comment */
```

will cause the compiler to dump core. 25 declarations on a line is a safe maximum. Break longer declarations into two or more statements to avoid this problem.

2.3 tty.h

In the file */usr/include/sys/tty.h* **t_proc** is an:

```
int (far *t_proc)()
```

When including this file, be sure to enable **near** and **far** keywords by using the **–Me** flag to **cc**.

2.4 types.h

Some of the C language *.h* files require that *<sys/types.h>* be included first. The error message:

```
old fashioned initialization
```

generally occurs when a type is used in an #include file but not declared. Try including *<sys/types.h>* earlier in the program.

2.5 uname(S)

No XENIX utilities currently use the node name in the **uname** structure. XENIX utilities use the entry in */etc/systemid*. However, some sites may want to set the node name in the **uname** structure. The node name can be up to 9 characters. You can set the node name by using the link kit to recompile a new XENIX kernel (refer to Chapter 10 of the XENIX *Operations Guide* "Installing Device Drivers"), or you can use **adb**(CP) to patch the existing XENIX kernel.

If you use **adb**, you must be logged in as super-user (root). Enter single-user, system maintenance mode (refer to **shutdown**(C)).

Consult the section entitled "uname (S)" of your "XENIX-86 and XENIX-286 Release 2.1.3 *Operating System Release Notes*" for the necessary **adb**(CP) commands to set the node name in the **uname**(S) structure.

3. Known Bugs

3.1 adb(CP) for XENIX –86

The program debugger **adb**(CP) works to patch binaries, runs on processes and *core* files, and sets breakpoints in XENIX-86. It does not perform stack backtraces. Subprocess control does not work on medium model programs. This will be fixed in a future release.

3.2 asx(CP)

The pre-cmerge assembler is included with this release for those users who have programs that require it. It is called */bin/asx* and is documented on the manual page **asx**(CP).

3.3 Floating–Point Exceptions

For compatibility reasons, floating-point exceptions (like dividing by zero) are masked within the **libc** libraries. Serious problems arise when **printf**(S) attempts to format the result of a "divide-by-zero". On an 80286 machine this gives a reasonable memory fault. On a 8086, an unmapped machine, anything may happen, from an infinite loop to a system crash.

4. Documentation Notes

This section discusses changes and errors in the documentation.

4.1 Improved Documentation

The following Development System chapters have been revised and expanded for this release:

Chapter 8– "Writing Device Drivers"– XENIX *C User's Guide*
Chapter 9– "Sample Device Drivers"– XENIX *C User's Guide*
Chapter 9– "Using System Resources"– XENIX *C Library Guide*

These are in the binder marked "Programmer's Guide Volume II".

4.2 cxref(CP)

cxref(CP) does not work in this release.

4.3 cc(CP)

Unless you have requested the beta **cmerge** utility from our support department, the **cc**(CP) and **ld**(CP) manual pages in the XENIX *Reference Manual* will not reflect the functionality of your release. Please replace them with the **cc**(CP) and **ld**(CP) manual pages included with these *Release Notes*. These are dated "December 18, 1985" and "August 26, 1985" respectively.

4.4 signal(S)

The *function address* value given as a *func* (prescribing the action **signal** is to execute) must be an even address. **Signal** treats odd address as **SIG_IGN** (causing the process to ignore the signal.)

5. Installation Notes

Note that you need to have the XENIX System V Release 2.1.3 (or equivalent) Operating System installed on your system in order to use the XENIX System V Release 2.1.3 Development System.

The XENIX Development System can be installed in several portions. If you intend to do DOS cross compilation, install the DOS development environment (linker, libraries and include files) in addition to the XENIX files. If you intend to do compilation for 286 machines, install those files. A complete list and short description of the sets included in the XENIX Development System, and all the files in each set, is available by using the **custom**(C) utility. **custom**(C) also informs you of the amount of space necessary to install each set. You can use **custom**(C) to install (or remove) all or part of the XENIX Development System at any time. Refer to the **custom**(C) manual page for instructions on using **custom**.

5.1 Packages In This Set

The Development System consists of the following packages:

Development System Packages

ALL	Entire development system set
PERM	XENIX contents and permissions lists
SOFT	Basic software development tools
LEX	Generates programs for lexical analysis
YACC	Yet another compiler-compiler
STACK	Determine program stack use (80286 only)
CREF	Generates cross reference listings
CFLOW	Generates C flow graphs
LINT	Syntax and usage check files and tools
MEDIUM	Medium Model Library routines
LARGE	Large model library routines
CCL	Large model compiler passes (80286 only)
SCCS	Source code control system
DOSDEV	DOS cross development libraries and utilities

Name

cc - Invokes the C compiler.

Syntax

cc [option ...] filename ...

Description

cc is the XENIX C compiler command. It creates executable programs by compiling and linking the files named by the *filename* arguments. *cc* copies the resulting program to the file **a.out**.

The *filename* can name any C or assembly language source file or any object or library file. C source files must have a **.c** filename extension. Assembly language source files must have **.s**, object files **.o**, and library files **.a** extensions. *cc* invokes the C compiler for each C source file and copies the result to an object file whose basename is the same as the source file but whose extension is **.o**. *cc* invokes the XENIX assembler, *masm* , for each assembly source file and copies the result to an object file with extension **.o**. *cc* ignores object and library files until all source files have been compiled or assembled. It then invokes the XENIX link editor, *ld* , and combines all the object files it has created together with object files and libraries given in the command line to form a single program.

Files are processed in the order they are encountered in the command line, so the order of files is important. Library files are examined only if functions referenced in previous files have not yet been defined. Library files must be in *ranlib*(CP) format, that is, the first member must be named __.SYMDEF, which is a dictionary for the library. The library is searched repeatedly to satisfy as many references as possible. Only those functions that define unresolved references are concatenated. A number of "standard" libraries are searched automatically. These libraries support the standard C library functions and program startup routines. Which libraries are used depends on the program's memory model (see "Memory Models" below). The entry point of the resulting program is set to the beginning of the "main" program function.

There are the following options:

-P
Preprocesses each source file and copies the result to a file whose basename is the same as the source but whose extension is **.i**. Preprocessing performs the actions specified by the preprocessing directives.

-E Preprocesses each source file as described for **-P**, but copies the result to the standard output. The option also places a #line directive with the current input line number and source file name at the beginning of output for each file.

-EP
Preprocesses each source file as described for **-E**, but does not place a #line directive at the beginning of the file.

-C
Preserves comments when preprocessing a file with **-E** or **-P**. That is, comments are not removed from the preprocessed source. This option may only be used in conjunction with **-E** or **-P**.

-D*name*[= *string*]
Defines *name* to the preprocessor as if defined by #define in each source file. The form "**-D***name*" sets *name* to 1. The form "**-D***name* = *string*" sets *name* to the given *string*.

-I *pathname*
Adds *pathname* to the list of directories to be searched when an #include file is not found in the directory containing the current source file or whenever angle brackets (< >) enclose the filename. If the file cannot be found in directories in this list, directories in a standard list are searched.

-X
Removes the standard directories from the list of directories to be searched for #include files.

-V *string*
Copies *string* to the object file created from the given source file. This option is often used for version control.

-W *num*
Sets the output level for compiler warning messages. If *num* is 0, no warning messages are issued. If 1, only warnings about program structure and overt type mismatches are issued. If 2, warnings about strong typing mismatches are issued. If 3, warnings for all automatic conversions are issued. This option does not affect compiler error message output.

-w Prevents compiler warning messages from being issued. Same as "-W 0".

-p
Adds code for program profiling. Profiling code counts the number of calls to each routine in the program and copies this information to the **mon.out** file. This file can be examined using the *prof*(CP) command.

-i
Creates separate instruction and data spaces for small model programs. When the output file is executed, the program text and data areas are allocated separate physical segments. The text portion will be read-only and may be shared by all users executing the file. This option is implied when creating middle or large model programs. (Not implemented on all machines.)

-F *num*
Sets the size of the program stack to *num* bytes. The value of *num* must be given in hexadecimal. The default stack for the 8086 is variable, starting at the top of a full 64 Kbyte data segment that grows down until it reaches data. The default stack for the 80286 is 2000 bytes (hexadecimal).

-Fs
Instructs the compiler to generate a source listing file.

-K
Removes stack probes from a program. Stack probes are used to detect stack overflow on entry to program routines.

-nl *num*
Sets the maximum length of external symbols to *num*. Names longer than *num* are truncated before being copied to the external symbol table.

-M *string*
Sets the program configuration. This configuration defines the program's memory model, word order, and data threshold. It also enables C language enhancements such as advanced instruction set and keywords. The *string* may be any combination of the following (the "s", "m", "l", and "h" are mutually exclusive):

s	Creates a small model program (default).
m	Creates a middle model program.
l	Creates a large model program.
h	Creates a huge model program.
e	Enables the far, near, huge, pascal, and fortran keywords.
0	Enables 8086 code generation for compiled C source files. Default is 8086 code generation.
1	Enables 186 code generation for compiled C source files.
2	Enables 286 code generation for compiled C source files.
b	Reverses the word order for **long** types. High order word is first. Default is low order word first.
t *num*	Sets the size of the largest data item in the data group to *num*. Default is 32,767.
d	Instructs the compiler not to assume SS=DS.

-c
Creates a linkable object file for each source file but does not link these files. No executable program is created.

-o *filename*
Defines *filename* to be the name of the final executable program. This option overrides the default name **a.out.**

-dos
Directs **cc** to create an executable program for MS-DOS systems.

-LARGE
Invokes the large model passes of the compiler (executable on 286 processors only). Using large model passes is advised when "Out of heap space" errors are encountered.

-S3
Causes the compiler to pass the **-v** *3* **-c** *2* flags to the link editor. This marks the executable *x.out* file created as a XENIX Version 3 file, runnable on a 286 cpu. It does not compile the program using 286 instructions (use -M2 for that).

-l*library*
Searches **/lib/lib***library***.a** for unresolved references to functions. The *library* must be an object file archive library in **ranlib** format.

-O
Invokes the object code optimizer.

-S
Creates an assembly source listing of the compiled C source file and copies this listing to the file whose basename is the same as the source but whose extension is **.s**. It should be noted that this file is not suitable for assembly. This option provides code for reading only.

-L Creates an assembler listing file containing assembled code and assembly source instructions. The listing is copied to the file whose basename is the same as the source but whose extension is **.L**. This option suppresses the -S option.

-NM *name*
Sets the module name for each compiled or assembled source file to *name*. If not given, the filename of each source file is used.

-NT *name*
Sets the text segment name for each compiled or assembled source file to *name*. If not given, the name "*module*_TEXT" is used for middle model and "_TEXT" for small model programs.

-ND *name*
Sets the data segment name for each compiled or assembled source file to *name*. If -ND is not given, the name "_DATA" is used.

Many options (or equivalent forms of these options) are passed to the link editor as the last phase of compilation. The -M option with the "s", "m", and "l" configuration options are passed to specify memory requirements. The -i, -F, and -p are passed to specify other characteristics of the final program.

The -D and -I options may be used several times on the command line. The -D option must not define the same name twice. These options affect subsequent source files only.

Memory Models

cc can create programs for four different memory models: small, middle, large, and huge. In addition, small model programs can be pure or impure.

Impure-Text Small Model
These programs occupy one 64K byte physical segment in which both text and data are combined. *cc* creates impure small model programs by default. They can also be created using the -Ms option.

Pure-Text Small Model
These programs occupy two 64K byte physical segments. Text and data are in separate segments. The text is read-only and may be shared by several processes at once. The maximum program size is 128 Kbytes. Pure small model programs are created using the -i and -Ms options.

Middle Model
These programs occupy several physical segments, but only one segment contains data. Text is divided among as many segments as required. Special call and returns are used to access functions in other segments. Text can be any size. Data must not exceed 64K bytes. Middle models programs are created using the -Mm option. These programs are always pure.

Large Model
These programs occupy several physical segments with both text and data in as many segments as required. Special calls and returns are used to access functions in other segments. Special addresses are used to access data in other segments. Text and data may be any size, but no data item may be larger than 64K bytes. Large model programs are created using the -Ml option. These programs are always pure.

Huge Model
These programs occupy several physical segments with both text and data in as many segments as required. It is possible to allow a data construct that spans 64K byte segments. This implementation imposes limits on the way the data construct is put together and where it is located in memory. Huge model programs are created using the **-Mh** option. These programs are always pure.

Small, middle, large and huge model object files can only be linked with object and library files of the same model. It is not possible to combine small, medium, large, and huge model object files in one executable program. *cc* automatically selects the correct small, middle, large, or huge versions of the standard libraries based on the configuration option. It is up to users to make sure that all of their own object files and private libraries are properly compiled in the appropriate model.

The special calls and returns used in middle, large, and huge model programs may affect execution time. In particular, the execution time of a program which makes heavy use of functions and function pointers may differ noticably from small model programs.

In middle, large, and huge model programs, function pointers are 32 bits long. In large and huge model programs, data pointers are 32 bits long. Programs making use of such pointers must be written carefully to avoid incorrect declaration and use of these variables.

The **-NM**, **-NT**, and **-ND** options may be used with middle, large, and huge model programs to direct the text and data of specific object files to named physical segments. All text having the same text segment name is placed in a single physical segment. Similarly, all data having the same data segment name is placed in a single physical segment.

Binary Compatibility

Programs created using this C compiler run on computers with several different Intel processors and versions of XENIX. Conversely, programs created on machines with different Intel processors and versions of XENIX can run on this version of XENIX. Refer to the manual page *machine*(M) for details.

Files

/bin/cc

See Also

ar(CP), ld(CP), lint(CP), machine(M), masm(CP), ranlib(CP)
XENIX *C User's Guide*, *C Library Guide*, and *C Language Reference*

Notes

Error messages are produced by the program that detects the error. These messages are usually produced by the C compiler, but may occasionally be produced by the assembler or the link loader.

All object module libraries must have a current *ranlib* directory. The user must make sure that the most recent library versions have been processed with *ranlib*(CP) before linking. If this is not done, *ld* cannot create executable programs using these libraries.

Name

ld – Invokes the link editor.

Syntax

ld [options] filename...

Description

ld is the XENIX link editor. It creates an executable program by combining one or more object files and copying the executable result to the file **a.out**. The *filename* must name an object or library file. By convention these names have the ".o" (for object) or ".a" (for archive library) extensions. If more than one name is given, the names must be separated by one or more spaces. If errors occur while linking, *ld* displays an error message; the resulting **a.out** file is unexecutable.

ld concatenates the contents of the given object files in the order given in the command line. Library files in the command line are examined only if there are unresolved external references encountered from previous object files. Library files must be in *ranlib*(CP) format, that is, the first member must be named __.SYMDEF, which is a dictionary for the library. *ld* ignores the modification dates of the library and the __.SYMDEF entry, so if object files have been added to the library since __.SYMDEF was created, the link may result in an "invalid object module."

The library is searched iteratively to satisfy as many references as possible and only those routines that define unresolved external references are concatenated. Object and library files are processed at the point they are encountered in the argument list, so the order of files in the command line is important. In general, all object files should be given before library files. *ld* sets the entry point of the resulting program to the beginning of the first routine.

ld should be invoked using the *cc*(CP) instead of invoking it directly. *cc* invokes *ld* as the last step of compilation, providing all the necessary C-language support routines. Invoking *ld* directly is not recommended since failure to give command line arguments in the correct order can result in errors.

There are the following options:

-A *num*
Creates a standalone program whose expected load address (in hexadecimal) is *num*. This option sets the absolute flag in the header of the a.out file. Such program files can only be executed as standalone programs. Options **-A** and **-F** are mutually exclusive.

-B *num*
Sets the text selector bias to the specified hexadecimal number.

-c *num*
Sets the target cpu. Default output is for 8086 processors. Specify **1** for 80186 processors or **2** for 80286. Using .286 instructions causes executable output suitable for 286 processors only, no matter what **-c** flag is specified.

-C
Causes the link editor to ignore the case of symbols.

-D *num*
Sets the data selector bias to the specified hexadecimal number.

-F *num*
Sets the size of the program stack to *num* bytes where *num is a hexadecimal number.* The default stack is a variable stack. Options **-F** and **-A** are mutually exclusive.

-i
Creates separate instruction and data spaces for small model programs. When the output file is executed, the program text and data areas are allocated separate physical segments. The text portion will be read-only and shared by all users executing the file.

-m name
Creates a link map file named *name* that includes public symbols.

-Ms
Creates small model program and checks for errors, such as fixup overflow. This option is reserved for object files compiled or assembled using the small model configuration. This is the default model if no **-M** option is given.

-Mm
Creates middle model program and checks for errors. This option is reserved for object files compiled or assembled using the middle model configuration. This option implies **-i**.

-Ml
Creates a large model program and checks for errors. The option is reserved for object files compiled using the large model configuration. This option implies **-i**.

-n *num*
Truncates symbols to the length specified by *num*.

-o *name*
Sets the executable program filename to *name* instead of **a.out**.

-P
Disables packing of segments

-s
Strips the symbol table.

-S *num*
Sets the maximum number of segments to *num*. If no argument is given, the default is 128.

-u *symbol*
Designates the specified *symbol* as undefined.

-v *num*
specifies the XENIX version number. Acceptable values for *num* are 2, 3, or 5; 5 is the default.

Files

/bin/ld

See Also

ar(CP), masm(CP), cc(CP), ranlib(CP)

Notes

The user must make sure that the most recent library versions have been processed with *ranlib*(CP) before linking. If this is not done, *ld* cannot create executable programs using these libraries.

5-1-86
SCO-014-030-003

Contents

Hardware Dependent (HW)

Name

intro – Introduction to machine related miscellaneous features and files.

Description

This section contains information useful in maintaining the system. Included are descriptions of files, devices, tables and programs that are important in maintaining the entire system that are directly related to the kind of computer the system runs on. This section is intended for use with the 86 family of Intel CPUs, specifically 8086, 8088, 80186, and 80286 based computers.

Name

boot – XENIX boot program.

Description

The *boot* program is an interactive program used to load and execute standalone XENIX programs. It is used primarily for loading and executing the XENIX kernel, but can be used for loading and executing any other programs that have been linked for standalone execution. The *boot* program is a required part of the XENIX Operating System and must be present in the root directory of the root file system to ensure successful loading of the XENIX kernel.

The *boot* program is invoked by the system each time the computer is started. For diskette boot, the procedure has two stages:

1. The boot block in sector 0 of the filesystem loads **/boot**.
2. **/boot** executes and prompts the user.

For fixed disk boot, the procedure has four stages:

1. The masterboot boot block in sector 0 loads the partition boot block from sector 0 of the active partition (see *fdisk*(C)).
2. Then, boot1 is loaded from the next three tracks.
3. boot1 loads **/boot**.
4. **/boot** executes and prompts the user.

/boot and **/xenix** may lie on tracks which have been mapped by *badtrk(M)* .

The fixed disk verison of *boot* is invoked if the diskette drive is empty and block 0 of the active partition of the fixed disk contains a valid bootstrap program.

When first invoked (or after termination of a standalone program), *boot* prompts for the location of a program to load by displaying the message:

```
XENIX System V

Boot
:
```

To specify the location of a program, a device and pathname must be given. The pathname must be the full pathname of the file containing the standalone program. You can display a list of the current allowable device names by typing the question mark (?).

The format for the device and pathname is as follows:

xx(m,o)filename

where:
xx=device name
('hd' for 'he hard disk or 'fd' for diskette device)
m =minor device number
(40 for the **root** filesystem)
o =offset in the partition (usually 0)

All numbers are in decimal. See the manual pages for *hd*(HW) and *fd*(HW) for minor device numbers of these devices.

For example, to load **xenix** from the diskette, enter:

fd(4,0)xenix

If the user has just loaded the *boot* program from the distribution diskette, simply press RETURN and *boot* defaults to the correct value.

To load XENIX from a hard disk, enter:

hd(40,0)xenix

If the user is loading both the *boot* program and XENIX from a hard disk, press RETURN when the system displays the boot message, and *boot* uses the default value, "hd(40,0)xenix ."

If nothing is typed after a short while, *boot* times out and acts as though a RETURN has been pressed. *boot* proceeds through the boot procedure, and *init*(M) is passed a **-a** flag.

It is recommended that DOS be installed on the hard disk before XENIX. This can be done by typing the word "dos" at the boot prompt:

Boot :dos

For more information on installing DOS see the manual page for fIdos (C).

Kernel Configuration

boot passes any boot string typed at the boot prompt to the kernel.

If the user types RETURN , *boot* assumes a default string.

The kernel reads the boot string to determine which peripherals are the root, pipe and swap. The default action is to choose the device from which the kernel is loaded.

Additional arguments in the boot string can alter this default action. These arguments have the form:

dev=xx(m,o)

Where:

dev = The desired system device (**rootdev**, **pipedev**, or **swapdev**)

xx, m, o = same as for the boot device

If only **rootdev** or **swapdev** is specified, then all system devices will be on that device, with the unspecified system devices using minor device numbers.

If both **rootdev** and **pipedev** are specified, the location of **swapdev** defaults to the device last specified in the boot string.

Setting only **swapdev** does not affect the defaults for the other system devices.

Diagnostics

On an error, **boot** displays an error mesage, then returns to its prompt. The following is a list of the most common messages:

bad magic number
The given file is not an executable program.

can't open <*pathname*>
The supplied pathname does not correspond to an existing file, or the device is unknown.

Stage 1 boot failure
The bootstrap loader cannot find or read **boot**. You must restart the computer and supply a file system disk with **boot** in the root directory.

not a directory
The specified area on the device does not contain a valid XENIX filesystem.

zero length directory
Although an otherwise valid filesystem was found, it contains a directory of apparently zero length. This most often occurs when a pre– System V XENIX filesystem (with incorrect, or incompatable word ordering) is in the specified area.

fload:read(*x***)=***y*
An attempted read of *x* bytes of the file returned only *y* bytes. This is probably due to a premature end-of-file. It could also be caused because of a corrupted file, or incorrect word ordering in the header.

Files

/boot
/etc/default/autoboot
/etc/masterboot
/etc/hdboot1

See Also

autoboot(M), badtrk(M), fd(HW), fdisk(M), hd(HW), init(M), sulogin(M)

Notes

The computer tries to boot off any diskette in the drive. If the diskette does not contain a valid bootstrap program, errors occur.

The **boot** program cannot be used to load programs that have not been linked for standalone execution. To create standalone programs, the **-A** option of the XENIX linker (*ld*(CP)) and special standalone libraries must be used.

Although standalone programs can operate in real or protected mode, they must not be large or huge model programs. Programs in real mode can use the input/output routines of the computer's startup ROM.

Name

cmos – Displays and sets the configuration data base.

Syntax

cmos [address [value]]

Description

The *cmos* command displays and/or sets the values in the CMOS configuration data base. This battery-powered data base stores configuration information about the computer that is used at power up to define the system hardware configuration and to direct boot procedures. The data base is 64 bytes long and is reserved for system operation. Refer to your computer hardware manual for more information.

The *cmos* command is typically used to alter the current hardware configuration when new devices are added to the system. When only *address* is given, the command displays the value at that address. If both *address* and a *value* are given, the command assigns the value to that address. If no arguments are given, the command displays the entire contents of the data base.

The CMOS configuration data base may also be examined and modified by reading from and writing to **/dev/cmos** file. Because successful system operation depends on correct configuration information, the data base should be modified by experienced system administrators only.

The computer manufacturer's diagnostic diskette should be run before setting the CMOS data base.

Files

/etc/cmos
/dev/cmos

Notes

Not all computers have a CMOS configuration data base. Some computers use switches on the main system board to configure the system. Refer to your computer hardware reference manual to determine whether you have a configuration data base.

Name

console, tty[02–*n*] – Computer screen

Syntax

```
#include <sys/console.h>
ioctl(fd, cmd, buf)
int fd, cmd;
char *buf;
```

Description

The **console** and **tty[02–*n*]** device files provide character I/O between the system and the computer screen and keyboard. Each file corresponds to a separate teletype device. The number of device files available, *n*, depends upon the amount of memory in the computer. The system displays the number of available screens during the boot process.

The console screens are modeled after a 25 line, 80 column ASCII terminal. Although the color graphics, enhanced graphics and professional graphics also support 40 column lines, XENIX does not.

The console is the default device for system error messages, and is the only teletype device open when in single user mode and during the system boot sequence.

To get to the next consecutive screen, enter **Ctrl–PrtSc** using the **Ctrl** key, and the **PrtSc** key. Any active screen may be selected by entering **alt–Fn**, where **Fn** is one of the function keys. **F1** refers to the system console screen (**/dev/console**).

The console is configurable via the *mapkey*(M) utilities, or at a lower level through *ioctl*(S).

Keyboard processing goes through two tables. The key mapping table maps keystrokes to either an ASCII value or a special function and the string table maps functions keys to ASCII strings. See *keyboard*(HW).

Access

fd must be a file descriptor open to the console.
cmd can be one of:

GIO_KEYMAP	Get keyboard mapping table from kernel
PIO_KEYMAP	Put keyboard mapping table into kernel
GIO_SCRNMAP	Get screen mapping table from kernel
PIO_SCRNMAP	Put screen mapping table to kernel
GIO_STRMAP	Get string key mapping table from kernel
PIO_STRMAP	Put string key mapping table to kernel

buf must be one of the following types: *keymap_t , scrnmap_t , or strmap_t* as defined in **<sys/console.h>** .

Refer to your computer hardware manual for information on scan codes generated by the keyboard and character ROM arrangement. Keyboard mapping is discussed in the XENIX Reference section *keyboard*(HW).

Screen Mapping (GIO_SCRNMAP, PIO_SCRNMAP)

The screen mapping table maps extended ASCII (8-bit) characters to ROM characters. It is an array [256] of char (typedef *scrnmap_t*) and is indexed by extended ASCII values. The value of the elements of the array are the ROM character to display.

For example the following will change the ASCII character '#' to be displayed as a English pound sign.

```
#include <sys/console.h>
change_pound()
{
scrnmap_t scrntab;
   /*
    * get screen mapping table of standard output
    */
   if(ioctl(0, GIO_SCRNMAP, scrntab) == -1)
   {
      perror("screenmap read");
      exit(-1);
   }
   /* 156 is the ROM value of English pound sign and 30 is the ASCII
    * value of '#'.
    */
   scrntab[30] = 156;
   if(ioctl(0, PIO_SCRNMAP, scrntab) == -1)
   {
      perror("screenmap write");
      exit(-1);
   }
}
```

Notes

ASCII characters are mapped to ROM characters via the screen map on a per screen basis. String key mapping is also on a per screen basis, but keyboard mapping is on a global basis. Only the super-user can use PIO_KEYMAP, otherwise the ioctl() call will fail with *errno* set to EACCES.

ASCII characters less than 32 do not go through the screen output mapping and thus can not be mapped to ROM characters.

Screen Attribute Sequences

The following character sequences are defined by ANSI X3.64-1979 and may be used to control and modify the screen display. Each **Pn** is replaced by the appropriate ASCII number (decimal) to produce the desired effect. The last column is for *termcap*(M) codes, where "n/a" means not applicable.

ANSI	Sequence	Action	Termcap Code
ED (Erase in Display)	ESC[Pn J	Erases all or part of a display. **Pn=0**: erases from active position to end of display. **Pn=1**: erases from the beginning of display to active position. **Pn=2**: erases entire display.	cd
EL (Erase in Line)	ESC[Pn K	Erases all or part of a line. **Pn=0**: erases from active position to end of line. **Pn=1**: erases from beginning of line to active position. **Pn=2**: erases entire line.	ce
ECH (Erase Character)	ESC[Pn X	Erases **Pn** characters	n/a
CBT (Cursor Backward Tabulation)	ESC[Pn Z	Moves active position back **Pn** tab stops.	bt
SU (Scroll Up)	ESC[Pn S	Scroll screen up **Pn** lines, introducing new blank lines at bottom.	sf
SD (Scroll Down)	ESC[Pn T	Scrolls screen down Pn lines, introducing new blank lines at top.	sr
CUP (Cursor Position)	ESC[P1;P2 H	Moves active position to location **P1** (vertical) and **P2** (horizontal).	cm

HVP (Horizontal & Vertical Position)	ESC[P1;P2 f	Moves active position to location **P1** (vertical) and **P2** (horizontal).	n/a
CUU (Cursor Up)	ESC[Pn A	Moves active position up **Pn** number of lines.	up (ku)
CUD (Cursor Down)	ESC[Pn B	Moves active position down **Pn** number of lines.	do (kd)
CUF (Cursor Forward)	ESC[Pn C	Moves active position **Pn** spaces to the right.	nd (kr)
CUB (Cursor Backward)	ESC[Pn D	Moves active position **Pn** spaces backward.	bs (kl)
HPA (Horizontal Position Absolute)	ESC[Pn ‘	Moves active position to column given by **Pn**.	n/a
HPR (Horizontal Position Relative)	ESC[Pn a	Moves active position **Pn** characters to the right.	n/a
VPA (Vertical Position Absolute)	ESC[Pn d	Moves active position to line given by **Pn**.	n/a
VPR (Vertical Position Relative)	ESC[Pn e	Moves active position down **Pn** number of lines.	n/a
IL (Insert Line)	ESC[Pn L	Inserts **Pn** new, blank lines.	al
ICH (Insert Character)	ESC[Pn @	Inserts **Pn** blank places for **Pn** characters.	ic

DL (Delete Line)	ESC[Pn M	Deletes **Pn** lines.	dl
DCH (Delete Character)	ESC[Pn P	Deletes **Pn** number of characters.	dc
CPL (Cursor to Previous Line)	ESC[Pn F	Moves active position to beginning of line, Pn lines up.	n/a
CNL (Cursor Next Line)	ESC[Pn E	Moves active position to beginning of line, Pn lines down.	n/a
SGR (Select Graphic Rendition)	ESC[0 m	Resets bold, blink, blank, underscore, and reverse. **Color**: Restores normal selected colors.	n/a
	ESC[1 m	Sets bold. **Color**: Sets intensity (changes *color* to *lt_color*).	n/a
	ESC[4m	Sets underscore. **Color**: No effect.	n/a
	ESC[5m	Sets blink. **Color**: Changes background *lt_color* to *color*; foreground blinks.	n/a
	ESC[7m	Sets reverse video. **Color**: Uses reverse selected colors.	so
	ESC[10m	Select primary font.	GE
	ESC[11m	Select first alternate font. Allows ASCII characters less than 32 to be displayed as ROM characters.	n/a

ESC[12m	Select second alternate font. Toggles high bit of extended ASCII code before displaying as ROM characters.	GS

The following color codes and sequences are defined by International Organization for Standardization ISO DP 6429.

C	Color
0	Black
1	Red
2	Green
3	Yellow
4	Blue
5	Magenta
6	Cyan
7	White

ISO	Sequence	Action	Termcap Code
SGR (Select Graphic Rendition)	ESC[3Cm	**Color**: Selects foreground color *C*.	n/a
	ESC[4Cm	**Color**: Selects background color *C*.	n/a
	ESC[8m	Sets blank (non-display).	n/a

The following color codes and sequences are additional control sequences.

Cn	Color	Cn	Color
0	Black	8	Grey
1	Blue	9	Lt. Blue
2	Green	10	Lt. Green
3	Cyan	11	Lt. Cyan
4	Red	12	Lt. Red
5	Magenta	13	Lt. Magenta
6	Brown	14	Yellow
7	White	15	Lt. White

Name	Sequence	Action	Termcap Code
SGR	ESC[2;C1;C2 m	Color only. Sets foreground (C1) and background (C2) colors.	n/a
SGR	ESC[7;C1;C2 m	Reverse video. **Color:** Sets foreground (C1) and background (C2) reverse video colors.	n/a
SGR	ESC[3;0 m	Color only. Clears blink bit.	n/a
SGR	ESC[3;1 m	Color only. Sets blink bit.	n/a
SGR	ESC[4 m	Underscores. **Color:** No effect	n/a
n/a	ESC[Pn g	Accesses alternate graphics set. Not the same as "graphics mode." Refer to your owner's manual for decimal/character codes **(Pn)** and possible output characters.	n/a

n/a	ESC Q Fn '*string* '	Define function key **Fn** with *string*. String delimiters ' and ' may be any character not in *string*. Function keys are numbered 0 through 9 (**F1** = 0, **F2** = 1, etc.).	n/a

A listing of the keyboard functions, codes, characters and escape sequences that are sent by each key, appear in the files:

/usr/lib/keyboard/keys
/usr/lib/keyboard/strings
/usr/lib/console/screens

Files

/dev/console

/dev/tty[02 –*n*]

/usr/lib/console/screens

/usr/lib/keyboard/keys

/usr/lib/keyboard/strings

See Also

keyboard(HW), termcap(M), mapkey(M), multiscreen(M), setcolor(C), setkey(M)

Name

fd - floppy devices

Description

The **fd** devices implement the XENIX interface with floppy disk drives. Typically, the *tar*(C), *cpio*(C) or *dd*(C) commands are used to read or write floppy disks. For instance,

```
tar tvf /dev/fd0
```

tabulates the contents of the floppy disk in drive 0 (zero).

The block special **fd** devices are also block-buffered. The floppy driver can read or write 512 bytes at a time using raw i/o. Note that block transfers are always a multiple of the 1K disk block size.

The floppy devices are named **/dev/fd0** and **/dev/fd1** (see Notes, below, for more information about device naming procedure).

The corresponding character special (raw) devices, **/dev/rfd0** and **/dev/rfd1,** afford direct, unbuffered transmission between the floppy and the user's read or write transfer address in the user's program.

For information about formatting, see *format*(C).

The minor device number determines what kind of physical device is attached to each device file (see Notes).

Files

/dev/fd0	/dev/rfd048ds8	/dev/rfd096ds15
/dev/fd1	/dev/rfd148ds8	/dev/rfd196ds15
/dev/rfd0	/dev/rfd048ds9	/dev/rfd096ds8
/dev/rfd1	/dev/rfd148ds9	/dev/rfd196ds8
		/dev/rfd048ss8
		/dev/rfd148ss9

Notes

When accessing the character special floppy devices, the user's buffer must begin on a word boundary. The count in a *read*(S), *write*(S), or *lseek*(S) call to a character special floppy device must be a multiple of 512 bytes.

Device names determine the particular drive and media configuration. The device names have the form: fd048ds9 Where: fd0 = drive number (0, 1, 2 or 3) 48 = number of disk tracks per inch (48

or 96) ds = single or double sided floppy (ss or ds) 9 = number of sectors on the floppy (8 or 9)

For instance, /dev/fd048ss9 indicates a 48 track per inch, single sided, 9 sector floppy disk device in drive 0.

The minor device numbers for floppy drives depend on the drive and media configuration. The most common are:

	48tpi				96tpi	
	ds/8	ds/9	ss/8	ss/9	ds/15	ds/8
Drive	Minor Device Number					
0	12	4	8	0	52	44
1	13	5	9	1	53	45
2	14	6	10	2	54	46
3	15	7	11	3	55	47

The scheme for creating minor device numbers is as follows. When interpreted as a binary number, each bit of the minor device number represents some aspect of the device/media configuration.

For example, the minor device number for **/dev/fd048ss8** is "8." Interpreted as a binary number, 8 is:

00001000

This is how each bit, or binary digit, is significant:

48tpi - 0 96tpi - 1	Sectors per Track		ss - 0 ds - 1	Drive	
32	16	8	4	2	1
0	0	1	0	0	0

Only the last six digits of the number are used in minor device identification. The first significant digit is the third from the left. In this example, the third digit from the left is zero, thus the device is 48tpi. The next two digits mean:

Bits		Sectors per Track
16	8	
0	0	9
0	1	8
1	0	15

The fourth digit tells whether the floppy is single sided (ss - 0) or double sided (ds - 1). The last two signify the drive number:

Bits		Drive Number
2	1	
0	0	0
0	1	1
1	0	2
1	1	3

Using this information, you can construct any minor device numbers you need.

It is not advisable to format a low density (48tpi) diskette on a high density (96tpi) floppy drive. Low density diskettes written on a high density drive should be read on high density drives. They may or may not be readable on a low density drive.

Use error-free floppy disks for best results on reading and writing.

Name

hd – Internal hard disk drive

Description

The files **hd00, hd01** through **hd04, hd0a** through **hd0d, root**, and **swap** provide block-buffered access to the primary hard disk. The corresponding files for a second hard disk are listed below.

root refers to the root file system; **swap** refers to the swap area; **hd00** is the entire disk; **hd01** through **hd04** are the four partitions recognized by DOS. Partition one is generally the XENIX partition. These special device files access the disks via the system's normal buffering mechanism and may be read and written without regard to the size of physical disk records.

The following are the names of the fixed disk partitions. Each partition can be accessed through a block interface, for example **/dev/hd01**, or through a character (raw) interface, for example **/dev/rhd01**.

Device File Names for Fixed Disks		
Disk 1	Disk 2	Partition
/dev/hd00 /dev/rhd00	/dev/hd10 /dev/rhd10	entire disk
/dev/hd01 /dev/rhd01	/dev/hd11 /dev/rhd11	first partition
/dev/hd02 /dev/rhd02	/dev/hd12 /dev/rhd12	second partition
/dev/hd03 /dev/rhd03	/dev/hd13 /dev/rhd13	third partition
/dev/hd04 /dev/rhd04	/dev/hd14 /dev/rhd14	fourth partition
/dev/hd0a /dev/rhd0a	/dev/hd1a /dev/rhd1a	active partition
/dev/hd0d /dev/rhd0d	/dev/hd1d /dev/rhd1d	DOS partition
/dev/u /dev/ru		
/dev/root /dev/rroot		root file system
/dev/swap /dev/rswap		swap area

Note that the last three file names do not exist for a second disk.

The device file names for DOS partitions function similarly to **/dev/hd?a .**

To access DOS partitions, specify letters such as "C:" or "D:" to indicate first or second partitions. The file **/etc/default/msdos** contains lines that assign a letter abbreviation for the DOS device name. Refer to *dos*(C).

The following table lists the minor device numbers for possible disk partitions. The minor device names for the raw devices are identical.

Minor Device Numbers			
Partition:	Minor Device Number:	Partition:	Minor Device Number:
hd00	0	hd10	64
hd01	15	hd11	79
hd02	23	hd12	87
hd03	31	hd13	95
hd04	39	hd14	103
hd0a	47	hd1a	111
hd0d	55	hd1d	119
root	40	u (user choice)	104
swap	41	u1	105
u (on 1st disk)	42	u2	106
recover	46		

Files

/dev/hd0a
/dev/rhd0a
/dev/hd0?
/dev/rhd0?
/dev/hd0d
/dev/rhd0d

/dev/hd1a
/dev/rhd1a
/dev/hd1?
/dev/rhd1?
/dev/hd1d
/dev/rhd1d

/dev/u
/dev/ru
/dev/root
/dev/rroot
/dev/swap
/dev/rswap

See Also

badtrk(C), divvy(C), dos(C), mkdev(C)

Diagnostics

The following messages may be printed on the console:

invalid fixed disk parameter table

and:

Error on Fixed Disk (minor *n*), blkno = *nnnnn*,
cmd=*nnnnn*, status=*nnnn*,
sector = *nnnnn*, Cylinder/head = *nnnnn*

Possible reasons for the first error include:

- The kernel is unable to get drive specifications, such as number of heads, cylinders, and sectors per track, from the disk controller ROM.
- Improper configuration.
- The disk is not turned on.
- The disk is not supported.

The second error specifies the following information:

- *blkno :* The XENIX block number within the device.
- *cmd :* The last command sent to the disk controller.
- *status :* The first byte of error status from the disk controller.
- *sector* and *Cylinder/head* specify the location of a possible flaw. This information is used with *badtrk*(M).

Notes

On the first disk, **hd00** denotes the entire disk and is used to access the master boot record and partition table. For the second disk, **hd10** denotes the entire disk and is used to access its partition table. Do not write to **hd10** and **hd00**.

Name

keyboard – The console keyboard

Description

The keyboard is used to enter data, switch screens, and send certain control signals to the computer. XENIX makes use of several particular keys and key combinations. These keys and key combinations have special names that are unique to the XENIX system, and may or may not correspond to the keytop labels on your keyboard. These keys are described later.

When you press a key, one of the following happens:

An ASCII value is entered

A string is sent to the computer.

A function is initiated.

The meaning of another key, or keys, is changed.

When a key is pressed (a keystroke), the keyboard sends a code to the computer, where the keyboard driver interprets the signals. The interpretation of key codes may be modified so that keys can function differently from their default actions.

There are three special occurances, or keystrokes:

Switch screens.

Send signals.

Change the value of previous character, characters or string.

Switching Screens (Multiscreen)

To get to the next consecutive screen, enter **Ctrl–PrtSc** using the **Ctrl** key, and the **PrtSc** key. Any active screen may be selected by entering **alt–F***n*, where **Fn** is one of the function keys. **F1** refers to the system console screen (**/dev/console**).

Signals

A signal affects some process or processes. Examples of signals are **Ctrl–d** (end of input, exits from shell), **Ctrl–** (quits a process), **Ctrl–s** (stop output to the screen), and **Ctrl–q** (resume sending

output).

Typically, characters are mapped to signals using *stty*(C). Only signals can be mapped using *stty*.

Altering Values

The actual code sent to the keyboard driver can be changed by using certain keys in combination. For example, the SHIFT key changes the ASCII values of the alphanumeric keys. Holding down the **Ctrl** key while pressing another key sends a control code (**Ctrl–d**, **Ctrl–s**, **Ctrl–q**, etc.).

Special Keys

To help you find the special keys, the following table shows which keys on a typical console correspond to XENIX system keys. In this table, a hyphen (-) between keys means 'hold down the first key while pressing the second.'

XENIX Name	Keytop	Action
INTR	Del	Stops current action and returns to the shell. This key is also called the RUB OUT or INTERRUPT key.
BACKSPACE	(left arrow)	Deletes the first character to the left of the cursor.
Ctrl-d	Ctrl-d	Signals the end of input from the keyboard; also exits current shell.
Ctrl-h	Ctrl-h	Deletes the first character to the left of the cursor. Also called the ERASE key.
Ctrl-q	Ctrl-q	Restarts printing after it has been stopped with Ctrl-s.
Ctrl-s	Ctrl-s	Suspends printing on the screen (does not stop the program).
Ctrl-u	Ctrl-u	Deletes all characters on the current line. Also called the KILL key.
Ctrl-\	Ctrl-\	Quits current command and creates a *core* file, if allowed. (Recommended for debugging only.)

ESCAPE	Esc	Special code for some programs. For example, changes from insert mode to command mode in the *vi*(C) text editor.
RETURN	(down-left arrow or ENTER)	Terminates a command line and initiates an action from the shell.
Fn	Fn	Function key *n*. F1-F12 are unshifted, F13-F14 are shifted F1-F12, F25-F36 are control F1-F12, and F37-F48 are ctrl-shift F1-F12. The remaining Fn keys (F49-F60) are on the on the number pad (unshifted). F49 - '7' F50 - '8' F51 - '9' F52 - '-' F53 - '4' F54 - '5' F55 - '6' F56 - '+' F57 - '1' F58 - '2' F59 - '3' F50 - '0'

Keyboard Mapping (GIO_KEYMAP, PIO_KEYMAP)

The syntax for keyboard mapping using *ioctl(S)* is:

```
#include <sys/console.h>
ioctl(fd, cmd, buf)
int fd, cmd;
char *buf;
```

The keyboard mapping table maps the functionality of the keys. It is an array of [111] by [8] chars (typedef *keymap_t*). The table is indexed with the system scancode as the first dimension and the software state of the key press (see scan code section below). as the second dimension.

If the value in the table is 0 through 127 then it is the ASCII code of that key press. If the value is 128 through 255 then it is some special function (see **<sys/console.h>**).

For example, the following will change the ASCII value returned by the key press "SHIFT - a" from 'A' to '@'.

```
#include <sys/console.h>
change_A()
{
keymap_t keytab;
   /*
    * get keyboard table for the standard
    * input
    */
   if(ioctl(0,GIO_KEYMAP, keytab) == -1)
   {
      perror("keymap read");
      exit(-1);
   }
   /*
    * 30 is the scancode for the 'a' key
    */
   keytab[30][SHIFT] = '@';
   if(ioctl(0,PIO_KEYMAP, keytab) == -1)
   {
      perror("keymap write");
      exit(-1);
   }
}
```

String key mapping (GIO_STRMAP, PIO_STRMAP)

The string mapping table is where the function keys are defined. It is an array of 256 bytes (typedef *strmap_t*) where null terminated strings can be put to redefine the function keys. The first null terminated string is assigned to the first string key and the second string to the second string key and so on. There is no limit on the length of any particular string as long as the whole table does not exceed 256 bytes, including nulls. Strings can be made null by the introduction of extra null characters.

The following is a list of default function key values:

Default Function Key Values

Key #	Function	Shift Function	Ctrl Function	Ctrl Shift Function
1	ESC[M	ESC[Y	ESC[k	ESC[w
2	ESC[N	ESC[Z	ESC[l	ESC[x
3	ESC[O	ESC[a	ESC[m	ESC[y
4	ESC[P	ESC[b	ESC[n	ESC[z
5	ESC[Q	ESC[c	ESC[o	ESC[@

6	ESC[R	ESC[d	ESC[p	ESC[[
7	ESC[S	ESC[e	ESC[q	ESC[\
8	ESC[T	ESC[f	ESC[r	ESC[]
9	ESC[U	ESC[g	ESC[s	ESC[^
10	ESC[V	ESC[h	ESC[t	ESC[_
11	ESC[W	ESC[i	ESC[u	ESC[`
12	ESC[X	ESC[j	ESC[v	ESC[{

Home	ESC[H
Up arrow	ESC[A
Page up	ESC[I
Left arrow	ESC[D
5	ESC[E
Right arrow	ESC[C
End	ESC[F
Down arrow	ESC[B
Page down	ESC[G
Insert	ESC[L

The following program, called **chstr**, when invoked from the command line as:

chstr 0 "cd /usr/sys"

will change function key "F1" from "ESC[E" to "cd /usr/sys".

```
#include <sys/console.h>

main(argc, argv)
int argc;
char *argv[ ];
{
    int keynum;
    char *newstr;
        if (argc != 3)
        {
        printf("Usage: chstr keynum string\n");
            exit(1);
        }
       keynum = atoi(argv[1]);
       newstr = argv[2];
       chstr(keynum, newstr);
}
```

```
/* F1 is string key #0 */

chstr(kn, ns)
int kn;
char *ns;
{
    strmap_t strtab;
    char *from, *to, *s=strtab;
    int i, oldlen, newlen, amount;

        if (ioctl(0, GIO_STRMAP, strtab) == -1)
        {
        perror("string table read failed");
            exit(1);
        }
    for(i=0; i<kn; i++) /* skip to right entry */
    while(*s++ != '\0')
            ;
        oldlen = strlen(s) + 1;
        newlen = strlen(ns)+ 1;
        if (oldlen > newlen)
        {    /* push table up for shorter string */
          from    = s + oldlen;
          to      = s + newlen;
          amount= STRTABLN-(s-strtab)-oldlen;
          while (--amount >= 0)
               *to++ = *from++;
          while (to < from)
               *to++ = '\0'; /* clear extra
                             table with NULL */
        }
        if (oldlen < newlen)
        {    /* push table down for longer string */
          from    = strtab+(STRTABLN-1)-(newlen-oldlen);
          to      = strtab+(STRTABLN-1);
          amount= STRTABLN-(s-strtab)-newlen;
               while(--amount >= 0)
               *to-- = *from--;
        }
        while( (*s++ = *ns++) != '\0')
          ;
        if (ioctl(0,PIO_STRMAP,strtab) == -1)
        {    perror("string table write failed");
          exit(1);
        }
}
```

Scan Codes

The following table is the default contents of **/usr/lib/keyboard/keys**. The column headings are:

SCAN CODE is the scan code generated by the keyboard hardware when a key is pressed. There is no user access to the scan code generated by releasing a key.

BASE is the normal value of a key press.

SHIFT is the value of a key press when the SHIFT is also being held down.

The other columns are the values of key presses when the CTRL, ALT and SHIFT keys are also held down.

All values greater than or equal to octal \200 are special key functions. All others are ASCII character values.

Special key functions are listed after the keyboard table.

SCAN CODE	BASE	SHIFT	CNTRL	CNTRL SHIFT	ALT	ALT SHIFT	ALT CNTRL	ALT CNTRL SHIFT
0	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'
1	'\033'	'\033'	'\377'	'\377'	'\033'	'\033'	'\377'	'\377'
2	'1'	'!'	'\377'	'\377'	'1'	'!'	'\377'	'\377'
3	'2'	'@'	'\000'	'\377'	'2'	'@'	'\000'	'\377'
4	'3'	'#'	'\377'	'\377'	'3'	'#'	'\377'	'\377'
5	'4'	'$'	'\377'	'\377'	'4'	'$'	'\377'	'\377'
6	'5'	'%'	'\377'	'\377'	'5'	'%'	'\377'	'\377'
7	'6'	'^'	'\036'	'\377'	'6'	'^'	'\036'	'\377'
8	'7'	'&'	'\377'	'\377'	'7'	'&'	'\377'	'\377'
9	'8'	'*'	'\377'	'\377'	'8'	'*'	'\377'	'\377'
10	'9'	'('	'\377'	'\377'	'9'	'('	'\377'	'\377'
11	'0'	')'	'\377'	'\377'	'0'	')'	'\377'	'\377'
12	'-'	'_'	'\037'	'\377'	'-'	'_'	'\037'	'\377'
13	'='	'+'	'\377'	'\377'	'='	'+'	'\377'	'\377'
14	'\b'	'\b'	'\177'	'\177'	'\b'	'\b'	'\177'	'\177'
15	'\t'	'\207'	'\377'	'\377'	'\t'	'\207'	'\377'	'\377'
16	'q'	'Q'	'\021'	'\377'	'q'	'Q'	'\021'	'\377'
17	'w'	'W'	'\027'	'\377'	'w'	'W'	'\027'	'\377'
18	'e'	'E'	'\005'	'\377'	'e'	'E'	'\005'	'\377'
19	'r'	'R'	'\022'	'\377'	'r'	'R'	'\022'	'\377'
20	't'	'T'	'\024'	'\377'	't'	'T'	'\024'	'\377'
21	'y'	'Y'	'\031'	'\377'	'y'	'Y'	'\031'	'\377'
22	'u'	'U'	'\025'	'\377'	'u'	'U'	'\025'	'\377'
23	'i'	'I'	'\t'	'\377'	'i'	'I'	'\t'	'\377'
24	'o'	'O'	'\017'	'\377'	'o'	'O'	'\017'	'\377'
25	'p'	'P'	'\020'	'\377'	'p'	'P'	'\020'	'\377'
26	'['	'{'	'\033'	'\377'	'['	'{'	'\033'	'\377'

27	']'	'}'	'\035'	'\377'	']'	'}'	'\035'	'\377'
28	'\r'	'\r'	'\n'	'\n'	'\r'	'\r'	'\n'	'\n'
29	'\201'	'\201'	'\201'	'\201'	'\201'	'\201'	'\201'	'\201'
30	'a'	'A'	'\001'	'\377'	'a'	'A'	'\001'	'\377'
31	's'	'S'	'\023'	'\377'	's'	'S'	'\023'	'\377'
32	'd'	'D'	'\004'	'\377'	'd'	'D'	'\004'	'\377'
33	'f'	'F'	'\006'	'\377'	'f'	'F'	'\006'	'\377'
34	'g'	'G'	'\007'	'\377'	'g'	'G'	'\007'	'\377'
35	'h'	'H'	'\b'	'\377'	'h'	'H'	'\b'	'\377'
36	'j'	'J'	'\n'	'\377'	'j'	'J'	'\n'	'\377'
37	'k'	'K'	'\013'	'\377'	'k'	'K'	'\013'	'\377'
38	'l'	'L'	'\f'	'\377'	'l'	'L'	'\f'	'\377'
39	';'	':'	'\377'	'\377'	';'	':'	'\377'	'\377'
40	'"'	'e"'	'\377'	'\377'	'e"'	'e"'	'\377'	'\377'
41	'`'	'~'	'\377'	'\377'	'`'	'~'	'\377'	'\377'
42	'\203'	'\203'	'\203'	'\203'	'\203'	'\203'	'\203'	'\203'
43	'\'	'|'	'\034'	'\034'	'\'	'|'	'\034'	'\034'
44	'z'	'Z'	'\032'	'\377'	'z'	'Z'	'\032'	'\377'
45	'x'	'X'	'\030'	'\377'	'x'	'X'	'\030'	'\377'
46	'c'	'C'	'\003'	'\377'	'c'	'C'	'\003'	'\377'
47	'v'	'V'	'\026'	'\377'	'v'	'V'	'\026'	'\377'
48	'b'	'B'	'\002'	'\377'	'b'	'B'	'\002'	'\377'
49	'n'	'N'	'\016'	'\377'	'n'	'N'	'\016'	'\377'
50	'm'	'M'	'\r'	'\377'	'm'	'M'	'\r'	'\377'
51	','	'<'	'\377'	'\377'	','	'<'	'\377'	'\377'
52	'.'	'>'	'\377'	'\377'	'.'	'>'	'\377'	'\377'
53	'/'	'?'	'\377'	'\377'	'/'	'?'	'\377'	'\377'
54	'\200'	'\200'	'\200'	'\200'	'\200'	'\200'	'\200'	'\200'
55	'*'	'*'	'\210'	'\210'	'*'	'*'	'\210'	'\210'
56	'\204'	'\204'	'\204'	'\204'	'\204'	'\204'	'\204'	'\204'
57	' '	' '	' '	' '	' '	' '	' '	' '
58	'\202'	'\202'	'\202'	'\202'	'\202'	'\202'	'\202'	'\202'
59	'\300'	'\314'	'\330'	'\344'	'\220'	'\220'	'\220'	'\220'
60	'\301'	'\315'	'\331'	'\345'	'\221'	'\221'	'\221'	'\221'
61	'\302'	'\316'	'\332'	'\346'	'\222'	'\222'	'\222'	'\222'
62	'\303'	'\317'	'\333'	'\347'	'\223'	'\223'	'\223'	'\223'
63	'\304'	'\320'	'\334'	'\350'	'\224'	'\224'	'\224'	'\224'
64	'\305'	'\321'	'\335'	'\351'	'\225'	'\225'	'\225'	'\225'
65	'\306'	'\322'	'\336'	'\352'	'\226'	'\226'	'\226'	'\226'
66	'\307'	'\323'	'\337'	'\353'	'\227'	'\227'	'\227'	'\227'
67	'\310'	'\324'	'\340'	'\354'	'\230'	'\230'	'\230'	'\230'
68	'\311'	'\325'	'\341'	'\355'	'\231'	'\231'	'\231'	'\231'
69	'\205'	'\205'	'\023'	'\023'	'\205'	'\205'	'\023'	'\023'
70	'\206'	'\206'	'\177'	'\177'	'\206'	'\206'	'\177'	'\177'
71	'\360'	'7'	'7'	'7'	'7'	'7'	'7'	'7'
72	'\361'	'8'	'8'	'8'	'8'	'8'	'8'	'8'
73	'\362'	'9'	'9'	'9'	'9'	'9'	'9'	'9'
74	'\363'	'-'	'-'	'-'	'-'	'-'	'-'	'-'
75	'\364'	'4'	'4'	'4'	'4'	'4'	'4'	'4'
76	'\365'	'5'	'5'	'5'	'5'	'5'	'5'	'5'
77	'\366'	'6'	'6'	'6'	'6'	'6'	'6'	'6'
78	'\367'	'+'	'+'	'+'	'+'	'+'	'+'	'+'

79	'\370'	'1'	'1'	'1'	'1'	'1'	'1'	'1'
80	'\371'	'2'	'2'	'2'	'2'	'2'	'2'	'2'
81	'\372'	'3'	'3'	'3'	'3'	'3'	'3'	'3'
82	'\373'	'0'	'0'	'0'	'0'	'0'	'0'	'0'
83	'\177'	'.'	'\177'	'\177'	'\177'	'\177'	'\177'	'\177'
84	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'
85	'\312'	'\326'	'\342'	'\356'	'\232'	'\232'	'\232'	'\232'
86	'\313'	'\327'	'\343'	'\357'	'\233'	'\233'	'\233'	'\233'
87	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'
88	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'
89	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'
90	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'
91	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'
92	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'
93	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'
94	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'
95	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'
96	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'
97	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'
98	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'
99	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'
100	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'
101	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'
102	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'
103	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'
104	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'
105	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'
106	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'
107	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'
108	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'
109	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'
110	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'
111	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'	'\377'

Special Key Values

Value	Meaning	Value	Meaning
\200	Right Shift	\201	Control
\202	Caps Lock	\203	Left Shift
\204	Alt	\205	Num Lock
\206	Scroll Lock	\207	Back Tab
\210	change to next screen		
\220	change to /dev/console	\221	change to /dev/tty02
\222	change to /dev/tty03	\223	change to /dev/tty04
\224	change to /dev/tty05	\225	change to /dev/tty06
\226	change to /dev/tty07	\261	change to /dev/tty08
\230	change to /dev/tty09	\231	change to /dev/tty10
\300	string key 0	\301	string key 1
\302	string key 2	\303	string key 3
\304	string key 4	\305	string key 5
\306	string key 6	\307	string key 7
\310	string key 8	\311	string key 9
	(continues through \373: string key 59)		
\373	string key 59	\377	No operation

Files

/usr/lib/keyboard/keys
/usr/lib/keyboard/strings

See Also

console(HW), mapkey(M), multiscreen(M), setkey(C), stty(C)

Name

lp, lp0, lp1, lp2 – Line printer device interfaces.

Description

The **lp**, **lp0**, **lp1**, and **lp2** files provide access to the optional parallel ports of the computer. The **lp0** and **lp2** files provide access to parallel ports 1 and 2, respectively. The **lp1** file provides access to the parallel port on the monochrome adaptor. Only one of **lp0** and **lp1** may be used on a given system. To access two parallel printers on a system, use either **lp0** or **lp1**, and **lp2**. The **lp** file is actually a link to either **lp0**, **lp1** or **lp2** and is the default output file for all *lp*(C) commands. Because the files are intended to give access to a standard dot matrix printer, all bytes written to a file are passed directly to the given port without translation.

Files

/dev/lp
/dev/lp0
/dev/lp1
/dev/lp2

See Also

lp(C), lpadmin(C), lpsched(C), lpinit(C)

Notes

The standard **lp** ports, **lp0**, **lp1**, and **lp2** send a printer initialization string the first time the file is opened after the system is *booted*.

Not all computers have an alternate parallel port slot.

Name

Machine – Description of host machine.

Description

This page lists the internal characteristics of personal computers which use the Intel 8086 processor family and its associated hardware. The information is intended for software developers who wish to transfer relocatable object or executable files from other XENIX machines to a personal computer then prepare the files for execution on the personal computer.

Central Processing Unit	Intel 8086, 8088, 80186, 80286
Disk Block Size (BSIZE)	1024 bytes
Memory Management Scheme	Unmapped (8086, 8088, 80186) Segmented (80286)
Split Instruction and Data	Supported
Variable Stack Size	Supported (8086 only) (8086 default configuration)
Fixed Stack Size	Supported (80286 default configuration)
Clock Ticks	.05 second (8086, 8088, 80186) .02 second (80286)

Binary Compatibility

The small and middle model binary programs created by the C compiler *cc*(CP) run on many processors. The following chart shows which XENIX systems running on which processors produce code executable on other machines. It is assumed that system specific system calls are not used to produce portable code. *cc*(CP) produces code by default, but can also be used as a cross development compiler, by using the appropriate flags.

SCO-*nn* is XENIX distributed by The Santa Cruz Operation, Inc. MS-*nn* is XENIX distributed by Microsoft Corporation. Intel XENIX is distributed by Intel Corporation. Altos XENIX is distributed by Altos Computer Systems. *nn* designates the machine processor. System designates the version of XENIX, either 2.3, 3.0, or System V.

Binary Compatibility			
Your System Processor	**Default compiler produces programs which run on System/Processor**	**Runs default programs created on System/Processor**	**Compiles (cross development) programs for System/Processor**
SCO-86 3.0	SCO-86 [3.0, Sys V] SCO-186 [3.0, Sys V] SCO-286 SysV	SCO-86 3.0 SCO-186 3.0 Intel, Altos-86 2.3, 3.0	DOS*
SCO-86 SystemV	SCO-86 Sys V SCO-186 Sys V SCO-286 Sys V MS-286 Sys V	SCO-86 [3.0, Sys V] SCO-186 [3.0, Sys V] SCO-286 Sys V Intel, Altos-86 2.3, 3.0	MS-286 3.0† DOS*
SCO-186 3.0	SCO-86 [3.0, Sys V] SCO-186 [3.0, Sys V] SCO-286 Sys V	SCO-86 3.0 SCO-186 3.0 Intel, Altos-86 2.3, 3.0	DOS*
SCO-186 System V	SCO-86 Sys V SCO-186 Sys V SCO-286 Sys V MS-286 Sys V	SCO-86 [3.0, Sys V] SCO-186 [3.0, Sys V] SCO-286 Sys V Intel, Altos-86 2.3, 3.0	MS-286 3.0† DOS*
SCO-286 3.0	SCO-286 [3.0, Sys V] MS-286 [3.0†, Sys V]	SCO-286 3.0 MS-286 3.0†	DOS*
SCO-286 System V	SCO-86 Sys V SCO-186 Sys V SCO-286 Sys V MS-286 Sys V	SCO-86 [3.0, Sys V] SCO-186 [3.0, Sys V] SCO-286 [3.0, Sys V] MS-286 [3.0†, Sys V]	SCO-286 3.0 MS-286 3.0† DOS*
MS-286 3.0†	MS-286 [3.0†, Sys V] SCO-286 Sys V	SCO-286 3.0	DOS*
MS-286 System V	MS-286 Sys V SCO-286 Sys V	SCO-86 [3.0, Sys V]‡ SCO-186 [3.0, Sys V]‡ SCO-286 [3.0, Sys V]‡	DOS*

* MS-DOS for i8086/8088, i80186 and i80286 processors.
† MS-286 3.0 XENIX is equivalent to Intel 286 3.0 XENIX.
‡ untested, pending release of this product.

See also

clockrate(M), cc(CP), ld(CP), a.out(F).

Name

parallel - Parallel interface devices.

Description

There are several parallel devices:

/dev/lp0 Main parallel adapter.

/dev/lp1 Adapter on monochrome video card.

/dev/lp2 Alternate parallel adapter (on appropriate machines).

/dev/lp is linked to the most often used parallel adapter interface. The minor device numbers are 0, 1 and 2 respectively.

It is not possible to have all three parallel devices on one system. 8086 computers only allow the use of **/dev/lp0**. Some 80286 computers allow the use of two parallel devices, either **/dev/lp0** or **/dev/lp1**, and **/dev/lp2**.

Notes

Parallel adapters on add-on cards will function, but switches must be set correctly.

Usage

Usually invoked by through *lp*(C), but can be written to directly.

Files

/dev/lp

See Also

lp(C), lp(HW), lpadmin(C), lpinit(C), lpsched(C), serial(HW)

Name

tty1[a-h] , tty1[A-H] , tty2[a-h] , tty2[A-H] – Interface to serial ports

Description

The **tty1[a–h]**, **tty1[A–H]**, **tty2[a–h]** and **tty2[A–H]** files provide access to the optional serial ports of the computer. Each file corresponds to one of the serial ports (with or without modem control). Files are named according to the following conventions:

- The first number in the file name corresponds to the COM expansion slot.
- Lower case letters indicate no modem control.
- Upper case letters indicate the line has modem control.

For example, the default serial lines are:

tty1a = 11
tty1A = 12
tty2a = 13
tty2A = 14

tty1a and **tty1A** both refer to COM 1, whereas **tty2a** and **tty2A** both refer to COM 2.

For example, with a four port expansion board installed at COM 1 and a single port board installed at COM 2, you can access:

tty1a tty1A
tty1b tty1B
tty1c tty1C
tty1d tty1D

tty2a tty2A

Each serial port has modem and non-modem invocations. The following device names refer to the serial ports, with and without modem control. The first section of the table describes boards at COM 1 and the second section describes boards installed at COM 2.

Serial Lines				
Board Type	Non-Modem Control		Modem Control	
	Minor	*Name*	*Minor*	*Name*
1 Port	0	tty1a	128	tty1A
4 Port	1	tty1b	129	tty1B
	2	tty1c	130	tty1C
	3	tty1d	131	tty1D
8 Port	4	tty1e	132	tty1E
	5	tty1f	133	tty1F
	6	tty1g	134	tty1G
	7	tty1h	135	tty1H
1 Port	8	tty2a	136	tty2A
4 Port	9	tty2b	137	tty2B
	10	tty2c	138	tty2C
	11	tty2d	139	tty2D
8 Port	12	tty2e	140	tty2E
	13	tty2f	141	tty2F
	14	tty2g	142	tty2G
	15	tty2h	143	tty2H

Interrupt Vectors:

All board(s) installed at COM 1 - 4
All board(s) installed at COM 2 - 3

For a list of I/O addresses, see the *Release Notes* furnished with your distribution.

Access

The files may only be accessed if the corresponding serial interface card is installed and its jumper I/O address correctly set. Also, for multi-port expansion cards, you must use the *mkdev (C)* program to create more than the default number of files. See *mkdev (C)* in the XENIX *Reference*.

The serial ports must also be defined in the system configuration. Check your hardware manual to determine how your system is configured, via a CMOS database or by switch settings on the main system board. If your system is configured using a CMOS database, the ports are defined in the database (see *cmos*(HW)). Otherwise, define the ports by setting the proper switches on the main system

board. Refer to your computer hardware manual for switch settings. It is an error to attempt to access a serial port that has not been installed and defined.

The serial ports can be used for a variety of serial communication purposes such as connecting login terminals to the computer, attaching printers, or forming a serial network with other computers. Note that a serial port may operate at most of the standard XENIX baud rates, and that the ports (on most computers) have a DTE (Data Terminal Equipment) configuration. The following table defines how each pin is used.

Pin	Description
2	Transmit Data
3	Receive Data
6	Request to Send
7	Signal Ground
8	Carrier Detect (Data Set Ready)
20	Data Terminal Ready

Only pins 2, 3, and 7 are necessary for a terminal (or direct) connection.

See *tty*(M) and *termio*(M) for the details of serial line operation in the XENIX system.

Files

/dev/tty1[a-h]
/dev/tty1[A-H]
/dev/tty2[a-h]
/dev/tty2[A-H]
/dev/tty11
/dev/tty12
/dev/tty13
/dev/tty14

See Also

cmos(HW), csh(C), cu(C), getty(M), mkdev(C), nohup(C), termio(M), tty(M), uucp(C)
XENIX *Installation Guide*

Notes

When using a serial line with *cu*(C) or *uucp*(C) modem control means that the line you log in on will be logged out when you hang up. Background processes may be killed. See *nohup*(C) and *csh*(C)). You must use a modem cable. Modem control should not be used for dial out situations.

Never attempt to use a port with both modem and non-modem control at the same time, or you see the warning:

"cannot open: device busy"

You cannot use the same serial port with both modem and non-modem control at the same time. For example, you cannot use, tty1a and tty1A simultaneously.

5-1-86
SCO-518-210-013

NOTES

NOTES